

# Basic use of the extension.

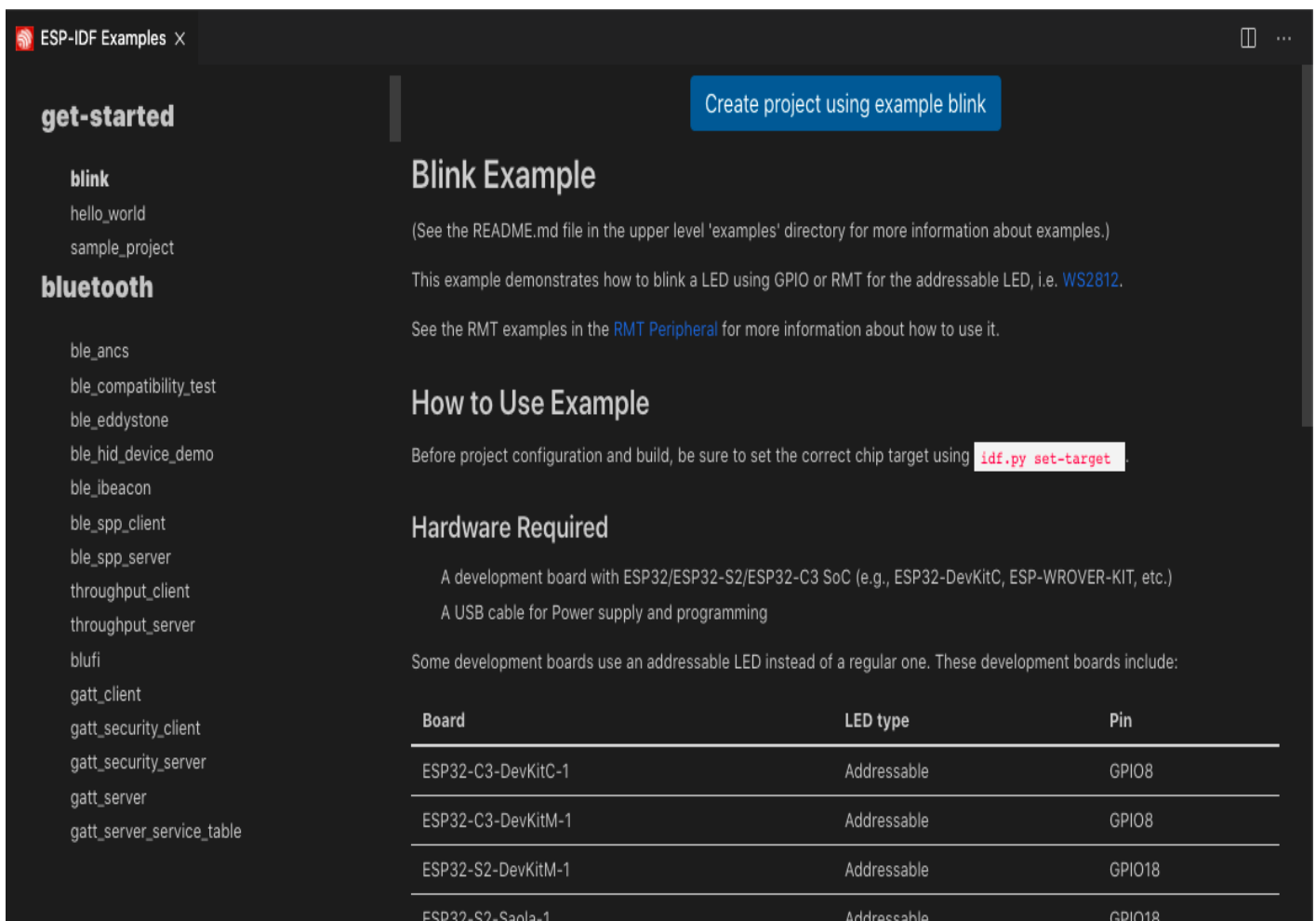
In this tutorial you will learn how to use the basic commands of this extension to develop your application with Espressif devices.

You have several options to create a project:

- Using one the examples from ESP-IDF or any additional supported framework using the **ESP-IDF: Show Examples Projects** command.
- Use one of the templates included with this extension using the **ESP-IDF: Create ESP-IDF project** command.

**NOTE:** To configure any additional supported framework, please review configuring additional frameworks

1. Let's use the ESP-IDF get-started's blink example for this tutorial. Click menu View -> Command Palette... and type **ESP-IDF: Show Examples Projects** and choose Use current ESP-IDF (/path/to/esp-idf) . If the user doesn't see the option, please review the setup in Install tutorial.
2. A window will be open with a list a projects, go the **get-started** section and choose the `blink_example` . You will see a **Create blink\_example project** button in the top and a description of the project below. Click **Create blink\_example project** button.



**ESP-IDF Examples** ×

**get-started**

**blink**

hello\_world

sample\_project

**bluetooth**

ble\_ancs

ble\_compatibility\_test

ble\_eddystone

ble\_hid\_device\_demo

ble\_ibeacon

ble\_spp\_client

ble\_spp\_server

throughput\_client

throughput\_server

blufi

gatt\_client

gatt\_security\_client

gatt\_security\_server

gatt\_server

gatt\_server\_service\_table

Create project using example blink

## Blink Example

(See the README.md file in the upper level 'examples' directory for more information about examples.)

This example demonstrates how to blink a LED using GPIO or RMT for the addressable LED, i.e. [WS2812](#).

See the RMT examples in the [RMT Peripheral](#) for more information about how to use it.

## How to Use Example

Before project configuration and build, be sure to set the correct chip target using `idf.py set-target`.

## Hardware Required

A development board with ESP32/ESP32-S2/ESP32-C3 SoC (e.g., ESP32-DevKitC, ESP-WROVER-KIT, etc.)

A USB cable for Power supply and programming

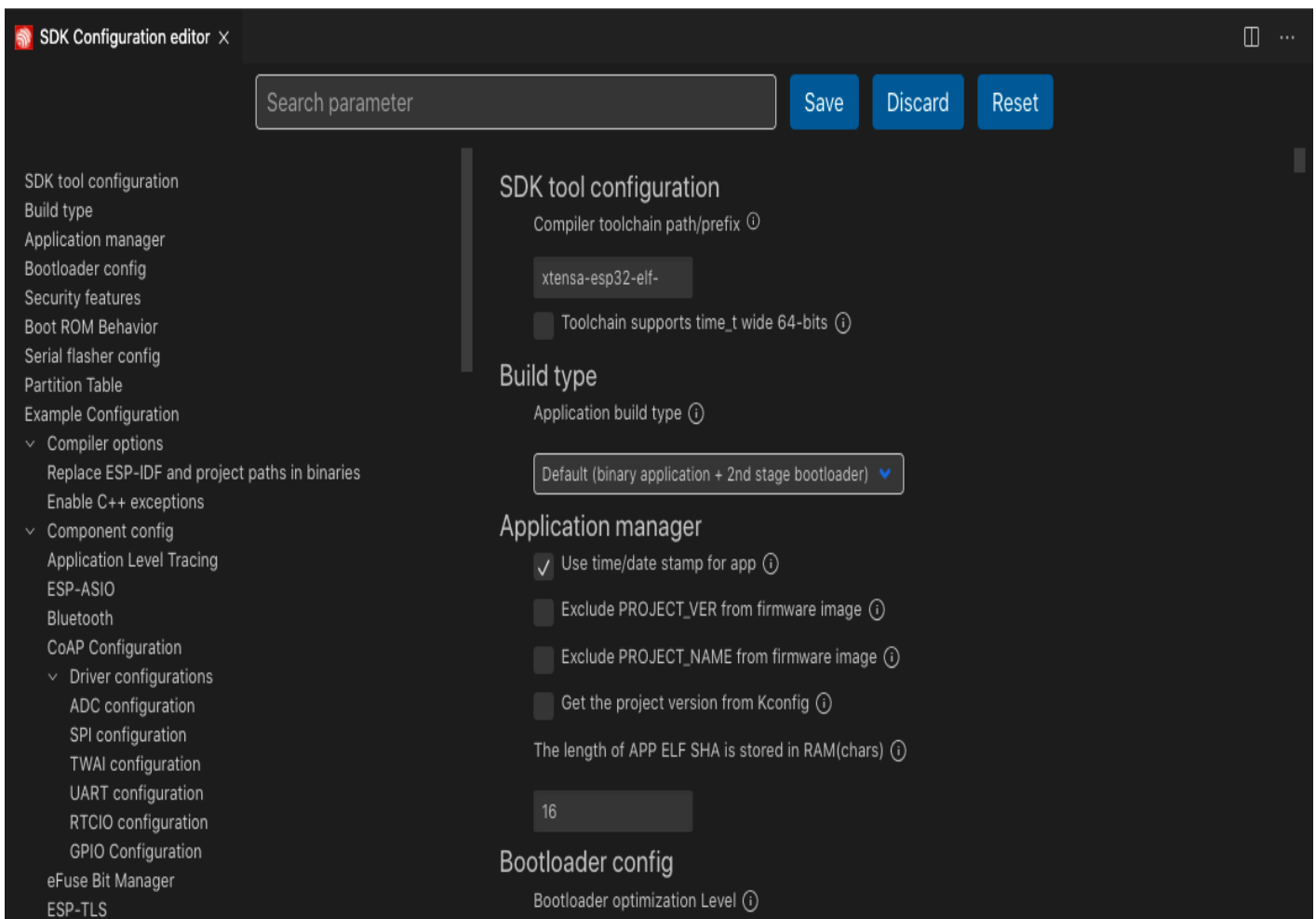
Some development boards use an addressable LED instead of a regular one. These development boards include:

Board	LED type	Pin
ESP32-C3-DevKitC-1	Addressable	GPIO8
ESP32-C3-DevKitM-1	Addressable	GPIO8
ESP32-S2-DevKitM-1	Addressable	GPIO18
ESP32-S2-Saola-1	Addressable	GPIO18

3. Now select a container directory where to copy the example project. For example, if the user choose `/Users/myUser/someFolder` the resulting folder will be `/Users/myUser/someFolder/blink` . This new project directory will be created and opened in Visual Studio Code.

4. First the user should select an Espressif target (esp32, esp32s2, etc.) with the **ESP-IDF: Set Espressif device target** command. Default is `esp32` and the one used in this tutorial.
5. Next configure your project using menuconfig. Use the **ESP-IDF: SDK Configuration editor** command ( `CTRL E G` keyboard shortcut ) where the user can modify the ESP-IDF project settings. After all changes are made, click save and close this window.

**NOTE:** The **SDK Configuration editor** is built from the project's `build/config/kconfig_menus.json` which is generated by the build system from ESP-IDF and user defined components `Kconfig` files on the first run of `SDK Configuration editor`. This process takes a bit of time so we keep the process running in the background to speed things up. If you are making changes to any `Kconfig` file or you want to re-run the `SDK Configuration editor` from scratch, you need to dispose the current process with the `ESP-IDF: Dispose current SDK Configuration editor server process` and run the `ESP-IDF: SDK Configuration editor` again.



6. Configure the `.vscode/c_cpp_properties.json` as explained in `C/C++ Configuration`.
7. Now to build the project, use the **ESP-IDF: Build your project** command ( `CTRL E B` keyboard shortcut). The user will see a new terminal being launched with the build output and a notification bar with Building Project message until it is done then a Build done message when finished. You could modify the behavior of the build task with `idf.cmakeCompilerArgs` for Cmake configure step and `idf.ninjaArgs` for Ninja step. For example, using `[-j N]` where `N` is the number of jobs run in parallel.

**NOTE:** There is a `idf.notificationSilentMode` configuration setting if the user does not wants to see the output automatically. Please review `ESP-IDF Settings`) to see how to modify this configuration setting.

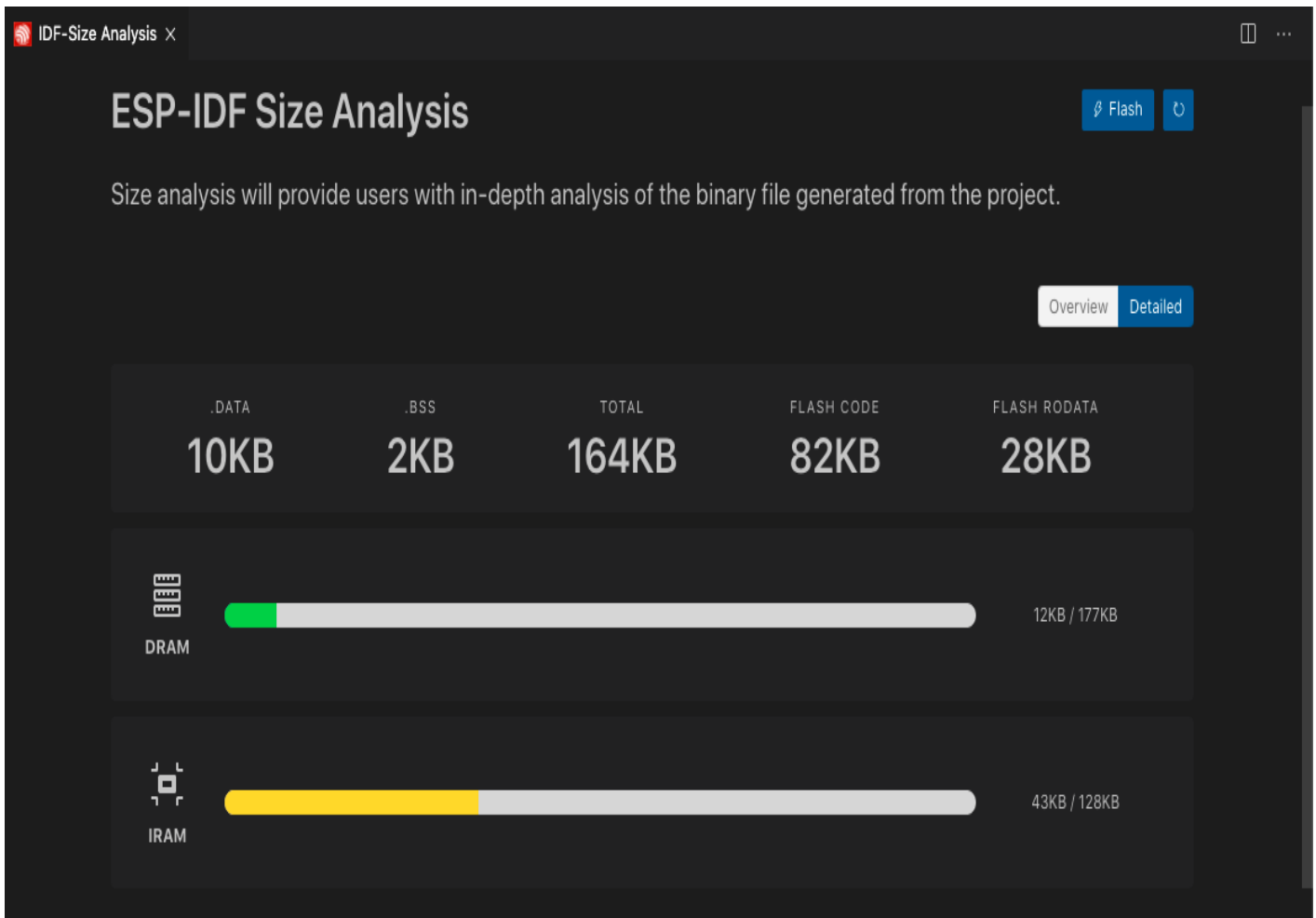
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Task
-- Components: bootloader bootloader_support efuse esp32 esp_common esp_hw_support esp_rom esp_system esptool_py hal log main micro-ecc newlib partition_table soc spi_flash xtensa
-- Component paths: /Users/brian/esp-idf/components/bootloader /Users/brian/esp-idf/components/bootloader_support /Users/brian/esp-idf/components/efuse /Users/brian/esp-idf/components/esp32 /Users/brian/esp-idf/components/esp_common /Users/brian/esp-idf/components/esp_hw_support /Users/brian/esp-idf/components/esp_rom /Users/brian/esp-idf/components/esp_system /Users/brian/esp-idf/components/esptool_py /Users/brian/esp-idf/components/hal /Users/brian/esp-idf/components/log /Users/brian/esp-idf/components/bootloader/subproject/main /Users/brian/esp-idf/components/bootloader/subproject/components/micro-ecc /Users/brian/esp-idf/components/newlib /Users/brian/esp-idf/components/partition_table /Users/brian/esp-idf/components/soc /Users/brian/esp-idf/components/spi_flash /Users/brian/esp-idf/components/xtensa
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/brian/workspace/blink/blink/build/bootloader
ninja: no work to do.
[4/4] Generating binary image from built executable
esptool.py v3.1-dev
Merged 1 ELF section
Generated /Users/brian/workspace/blink/blink/build/blink.bin

Terminal will be reused by tasks, press any key to close it.
Build Successfully

```

8. (OPTIONAL) Use the **ESP-IDF: Size analysis of the binaries** command ( `CTRL E S` keyboard shortcut) to review the application size information.



8. Before flashing the project, the user needs to specify the serial port of the device with the **ESP-IDF: Select port to use** command ( `CTRL E P` keyboard shortcut). You can choose between UART/JTAG flashing mode and then a list of serial ports will be shown for the user to select.

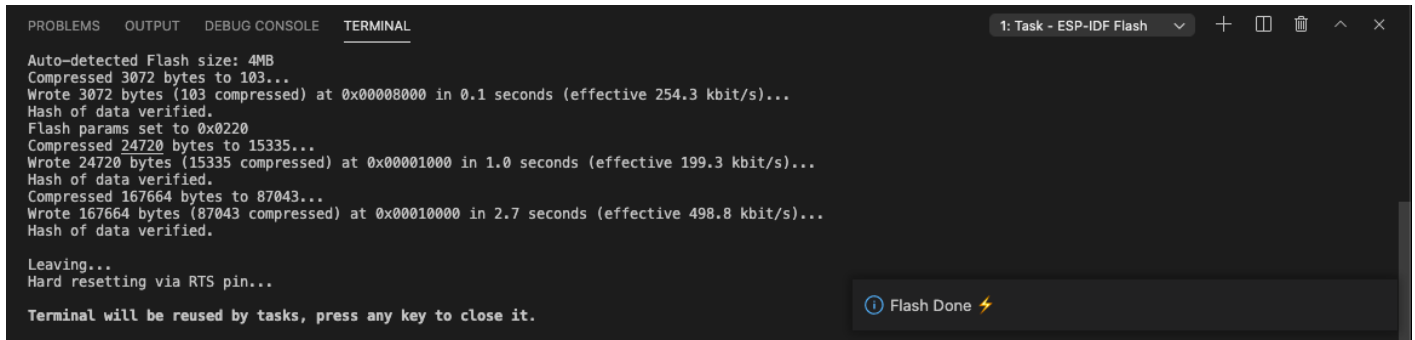
**NOTE:** Please take a look at ESP-PROG board the instructions or Configuring ESP32 Target your Espressif device and JTAG interface to your computer.

9. Now to flash the project, use the **ESP-IDF: Flash your project** command ( `CTRL E F` keyboard shortcut). Choose `UART` or `JTAG` flash mode (Configure JTAG flashing) and then flashing will start in the previously selected serial port. The user can also use the **ESP-IDF: Flash (UART) your project** or **ESP-IDF: Flash (with JTAG)** directly.

**NOTE:** When using the **ESP-IDF: Select Flash Method and Flash** command, your choice will be saved in the `idf.flashType` configuration setting.

The user will see a new terminal being launched with the flash output and a notification bar with **Flashing Project** message until it is done then a **Flash done** message when finished.

**NOTE:** There is an `idf.flashBaudRate` configuration settings to modify the flashing baud rate. Please review **ESP-IDF Settings** to see how to modify this configuration setting.



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Task - ESP-IDF Flash

Auto-detected Flash size: 4MB
Compressed 3072 bytes to 103...
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effective 254.3 kbit/s)...
Hash of data verified.
Flash params set to 0x0220
Compressed 24720 bytes to 15335...
Wrote 24720 bytes (15335 compressed) at 0x00001000 in 1.0 seconds (effective 199.3 kbit/s)...
Hash of data verified.
Compressed 167664 bytes to 87043...
Wrote 167664 bytes (87043 compressed) at 0x00010000 in 2.7 seconds (effective 498.8 kbit/s)...
Hash of data verified.

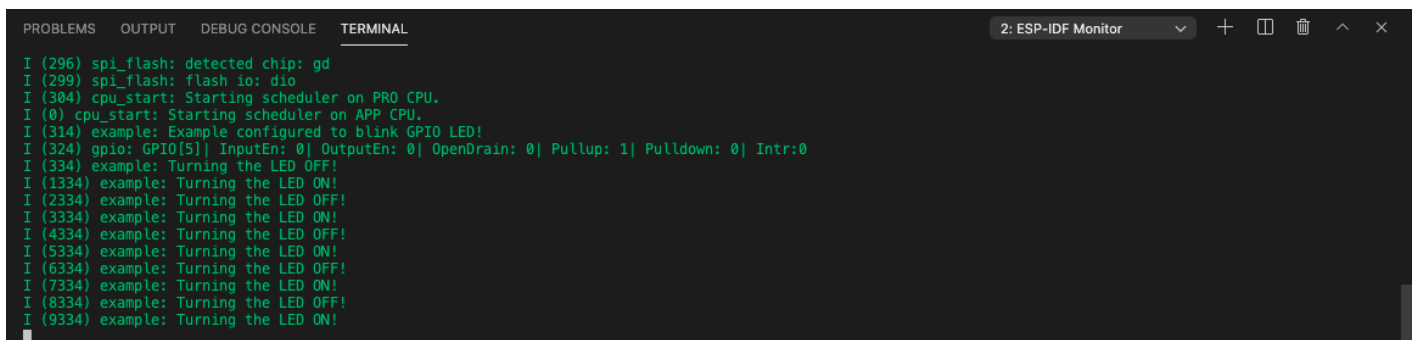
Leaving...
Hard resetting via RTS pin...

Terminal will be reused by tasks, press any key to close it.
Flash Done ⚡

```

10. Now to start monitoring your device, use the **ESP-IDF: Monitor your device** command ( `CTRL E M` keyboard shortcut). The user will see a new terminal being launched with the `idf.py monitor` output.

**NOTE** The ESP-IDF Monitor baud rate value is taken from `idf.monitorBaudRate` with fallback on your project's `skdconfig CONFIG_ESPTOOLPY_MONITOR_BAUD` (`idf.py monitor` baud rate). This value can also be override by setting the environment variable `IDF_MONITOR_BAUD` or `MONITORBAUD` in your system environment variables or this extension's `idf.customExtraVars` configuration setting. Please review **ESP-IDF Settings** to see how to modify `idf.customExtraVars`.



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: ESP-IDF Monitor

I (296) spi_flash: detected chip: gd
I (299) spi_flash: flash io: dio
I (304) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
I (314) example: Example configured to blink GPIO LED!
I (324) gpio: GPIO[5] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 1 Pulldown: 0 Intr:0
I (334) example: Turning the LED OFF!
I (1334) example: Turning the LED ON!
I (2334) example: Turning the LED OFF!
I (3334) example: Turning the LED ON!
I (4334) example: Turning the LED OFF!
I (5334) example: Turning the LED ON!
I (6334) example: Turning the LED OFF!
I (7334) example: Turning the LED ON!
I (8334) example: Turning the LED OFF!
I (9334) example: Turning the LED ON!

```

## Next steps

You can debug ESP-IDF projects as shown in the debug tutorial.

The **ESP-IDF: Open ESP-IDF Terminal** will launch a system terminal with ESP-IDF, ESP-IDF tools and ESP-IDF python virtual environment loaded as environment variables. Just typing `idf.py` or `esptool.py` should work to execute scripts from ESP-IDF and additional frameworks.

See other ESP-IDF extension features.

## About JTAG flashing

JTAG flash mode requires openOCD v0.10.0-esp32-20201125 or later. To replace openOCD, just get one of the latest openOCD releases and replace in `idf.customExtraPaths` the openOCD binary path like:

```
c:\\esp\\tools\\.espressif\\tools\\openocd-esp32\\v0.10.0-esp32-  
20200709\\openocd-esp32\\bin
```



for the bin directory of your desired openOCD release

```
c:\\esp\\tools\\.espressif\\tools\\openocd-esp32\\v0.10.0-esp32-  
20201202\\openocd-esp32\\bin
```



Also update `idf.customExtraVars` `OPENOCD_SCRIPTS` to the new OpenOCD Scripts folder path.

Please review ESP-IDF Settings to see how to modify these values.